

Элементы технического зрения ПРТС

БИБЛИОТЕКА EIGEN. ИТЕРАЦИОННЫЕ МЕТОДЫ. МЕТОД
НАИМЕНЬШИХ КВАДРАТОВ.

Eigen – чуть подробнее

На предыдущем занятии мы выяснили, как

- Создавать вектора и матрицы в библиотеке Eigen
- С помощью библиотеки найти решение, используя LU-разложение
- Теперь попробуем посмотреть чуть подробнее

Eigen – LU- разложение

```
#include <iostream>
#include <eigen3/Eigen/Core>
#include <eigen3/Eigen/LU>

using namespace std;

int main(int argc, char *argv[])
{
    Eigen::Matrix4d A, lower, upper;
    Eigen::Vector4d b, x;

    A << 10, 6, 2, 0, 5, 1, -2, 4, 3, 5, 1, -1, 0, 6, -2, 2;
    b << 25, 14, 10, 8;

    Eigen::PartialPivLU<Eigen::Matrix4d> lu(A);

    cout << "Matrix A" << endl << A << endl;
    cout << "Vector b" << endl << b << endl;
    cout << "LU-decomp" << endl << lu.matrixLU() << endl;

    lower = lu.matrixLU().triangularView<Eigen::StrictlyLower>();
    lower = lower + Eigen::Matrix4d::Identity();
    upper = lu.matrixLU().triangularView<Eigen::Upper>();

    cout << "Matrix lower" << endl << lower << endl;
    cout << "Matrix upper" << endl << upper << endl;

    cout << "Multiplication" << endl << lower*upper << endl;
    cout << "Multiplication with permutation" << endl
        << lu.permutationP().inverse()*lower*upper << endl;

    return 0;
}
```

Eigen – LU- разложение

```
Matrix A
10  6  2  0
 5  1 -2  4
 3  5  1 -1
 0  6 -2  2
Vector b
25
14
10
 8
LU-decomp
      10      6      2      0
      0      6     -2      2
      0.5 -0.333333 -3.66667  4.66667
      0.3  0.533333  -0.4    -0.2
Matrix lower
      1      0      0      0
      0      1      0      0
      0.5 -0.333333      1      0
      0.3  0.533333     -0.4      1
Matrix upper
      10      6      2      0
      0      6     -2      2
      0      0 -3.66667  4.66667
      0      0      0    -0.2
Multiplication
10  6  2  0
 0  6 -2  2
 5  1 -2  4
 3  5  1 -1
Multiplication with permutation
10  6  2  0
 5  1 -2  4
 3  5  1 -1
 0  6 -2  2
```

Eigen – Различные методы решения

FullPivLU – LU-разложение с полным выбором главного элемента

PartialPivLU - LU-разложение с выбором главного элемента по строкам

HouseholderQR – QR-разложение

LLT – метод Холецкого

...

А.А. Амосов, Ю.А. Дубинский, Н.В. Копчёнова «Вычислительные методы для инженеров. Учебное пособие» - М., Высшая школа, 1994г.

Eigen – Различные методы решения

```
1  #include <iostream>
2  #define _USE_PREDEFINED_CONSTANT
3  #include <cmath>
4
5  #include <eigen3/Eigen/LU>
6  #include <eigen3/Eigen/QR>
7  #include <eigen3/Eigen/SVD>
8  #include <eigen3/Eigen/Dense>
9  #include <eigen3/Eigen/Core>
10 using std::cout;
11 using std::endl;
12 using Eigen::MatrixXf;
13 using Eigen::VectorXf;
14
15 #define SYS_SIZE 3
16
17 int main()
18 {
19     MatrixXf a(SYS_SIZE, SYS_SIZE);
20
21     a << 8.1041, 0.0083, -3.1433, 1.7114, -0.0200, -0.6638, 4.2800, -0.0078, -1.6600;
22     // a = Eigen::MatrixXf::Random(SYS_SIZE, SYS_SIZE);
23     cout << "Matrix A\n" << a << endl;
24     VectorXf b(SYS_SIZE), xplu(SYS_SIZE), xflu(SYS_SIZE), xqr(SYS_SIZE);
25     b << -4, 6.00, -8.5;
26     // b = Eigen::VectorXf::Random(SYS_SIZE);
27     Eigen::PartialPivLU<MatrixXf> plu(a);
28     xplu = plu.solve(b);
29     cout << "Partial pivoting solution\n" << xplu << endl;
30     Eigen::FullPivLU<MatrixXf> flu(a);
31     xflu = flu.solve(b);
32     cout << "Full pivoting solution\n" << xflu << endl;
33     Eigen::FullPivHouseholderQR<MatrixXf> qr(a);
34     xqr = qr.solve(b);
35     cout << "QR solution\n" << xqr << endl;
36
37     cout << "Difference\n" << xplu - xqr << endl;
38     cout << "For partially pivoted LU Ax-b=\n" << a*xplu-b << endl;
39     cout << "For QR Ax-b=\n" << a*xqr-b << endl;
40
41     return 0;
42 }
```

Eigen – Различные методы решения

```
Matrix A
 8.1041  0.0083 -3.1433
 1.7114  -0.02 -0.6638
  4.28 -0.0078  -1.66
Partial pivoting solution
-58282.2
-265.251
-150263
Full pivoting solution
-58282.2
-265.251
-150263
QR solution
-58307.2
-265.329
-150328
Difference
 25.0078
 0.0785522
 64.4844
For partially pivoted LU Ax-b=
 0
-0.0078125
-0.015625
For QR Ax-b=
 0.03125
 0
-0.015625
Press <RETURN> to close this window...
```

Итерационные методы решения СЛАУ

- Вместо точного решения мы ищем приближённое

- Мы преобразуем исходную задачу к специальному виду:

$$Ax = b \Rightarrow \tilde{A}x = Bx + b \Rightarrow x = \tilde{A}^{-1}Bx + \tilde{A}^{-1}b \Rightarrow x = \tilde{B}x + \tilde{b}$$

где

$$A = \tilde{A} + B, \tilde{A} = \text{diag}(A)$$

- Тогда, если мы зададим начальное приближение x_0 , получим итерационный процесс

$$x_{n+1} = \tilde{A}^{-1}Bx_n + \tilde{A}^{-1}\tilde{b}$$

- Метод простой итерации

Сходимость итерационных методов

- Всегда ли итерационные методы сходятся?

- Сходимость основана на свойстве сжимающих отображений

$$\rho(Ax, Ay) < \alpha \rho(x, y), 0 \leq \alpha < 1$$

- Если отображение сжимающее, то метод сходится
 - Для каждого итерационного метода этот критерий свой
 - Например, для метода простой итерации $\|\tilde{B}x\| \leq \|x\|$

- Но, вот беда, мы не знаем x , как тогда проверить выполнение этого условия?

- Можно построить оценку для произвольного элемента x , если воспользоваться согласованной нормой матрицы

$$\|Ax\| \leq \|A\| \cdot \|x\|$$

- Но тогда встаёт вопрос – что же такое норма?

Нормы векторов и матриц

- Мы будем понимать под нормой вектора его длину
- Под нормой матрицы мы будем подразумевать операторную норму

$$\|A\| = \max_{\|x\|=1} \|Ax\|$$

- Получается, что норма матрицы зависит от способа вычисления нормы вектора
- Используются т.н. нормы Гёльдера для векторов:

$$\|x\|_{l^p} = \left(\sum_{i=1}^n x_i^p \right)^{\frac{1}{p}}$$

- Зачем?

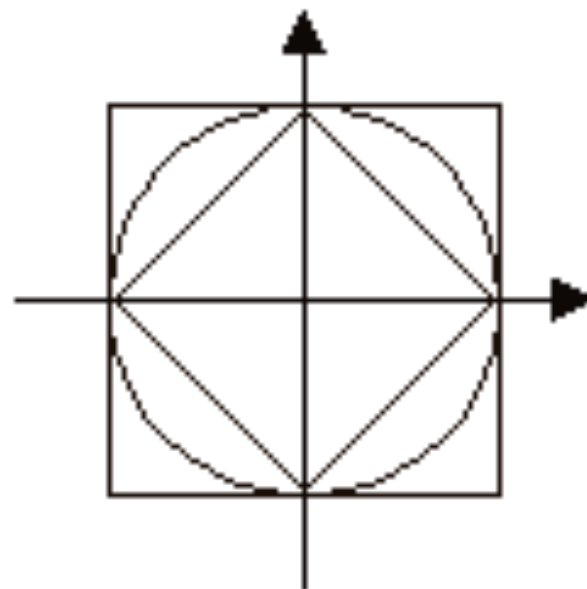
Согласованные нормы

- Частные, наиболее распространённые случаи:

$$\|x\|_1 = \sum_i x_i \Rightarrow \|A\|_1 = \max_j \sum_i |a_{ij}|$$

$$\|x\|_\infty = \max_i x_i \Rightarrow \|A\|_\infty = \max_i \sum_j |a_{ij}|$$

$$\|x\|_2 = \sqrt{\sum_i x_i^2} \Rightarrow \|A\|_2 = \sqrt{\lambda_{\max}(A^T A)} \quad (Ax, Ax) = \lambda_i$$



Реализация метода простой итерации

```
#include <iostream>
#include <eigen3/Eigen/Dense>

using std::cout;
using std::endl;

int main(int argc, char *argv[])
{
    Eigen::Matrix3d mat, A, B;
    mat << 6.25, -1, 0.5, \
          -1, 5, 2.12, \
          0.5, 2.12, 3.6;
    cout << mat << endl;
    Eigen::Vector3d b, b_orig, x, x_prev=Eigen::Vector3d::Zero();
    b << 7.5, -8.68, -0.24;
    b_orig = b;
    A = mat.diagonal().asDiagonal();
    B = -mat + A;
    for (int i=0; i<B.rows(); i++)
    {
        b(i)/=A(i,i);
        B.row(i) /= A(i,i);
    }
}
```

Реализация метода простой итерации

Для вычисления других норм
используется функция

`lpNorm<n>()`

`lpNorm<Eigen::Infinity>()`

```
cout << "B=\n" << B << endl;
cout << "Norm of B=" << B.norm() << endl;
double normX=x.norm(), normXprev, tolerance = 0.0000001;
int counter = 0;

do
{
    x = B*x_prev+b;
    counter++;
    normXprev = normX;
    normX = x.norm();
    x_prev = x;
}
while(fabs(normX-normXprev)> tolerance);

cout << "After " << counter << " iterations we found X\n" << x << endl;
Eigen::Vector3d residual = mat*x-b_orig;
cout << "Residual norm " << std::fixed << residual.norm() << endl;
Eigen::Vector3d xLU = mat.lu().solve(b_orig);
cout << "Same thing using LU-decomposition\n" << xLU<< endl;

return 0;
}
```

Результат работы программы

```
Starting /Users/amakashov/projects/build-linsolve-Desktop-Debug/linsolve...
6.25  -1  0.5
  -1   5 2.12
  0.5 2.12  3.6
B=
      0      0.16    -0.08
     0.2      0    -0.424
-0.138889 -0.588889      0
Norm of B=0.786038
After 29 iterations we found X
0.8
-2
1
Residual norm 0.000000
Same thing using LU-decomposition
0.800000
-2.000000
1.000000
/Users/amakashov/projects/build-linsolve-Desktop-Debug/linsolve exited with code 0
```

Достоинства и недостатки

- + При большом размере матрицы позволяют находить решение быстрее прямых
- + В них меньше накапливается вычислительная погрешность
- + Позволяют «управлять» точностью получаемого решения
- Не всегда применимы
- Для матриц небольшого размера – нерационально использовать
- В некоторых случаях – требуется найти «хорошее» приближение

```
Eigen::Matrix3d low, up;
```

```
low = B.triangularView<Eigen::StrictlyLower>();
```

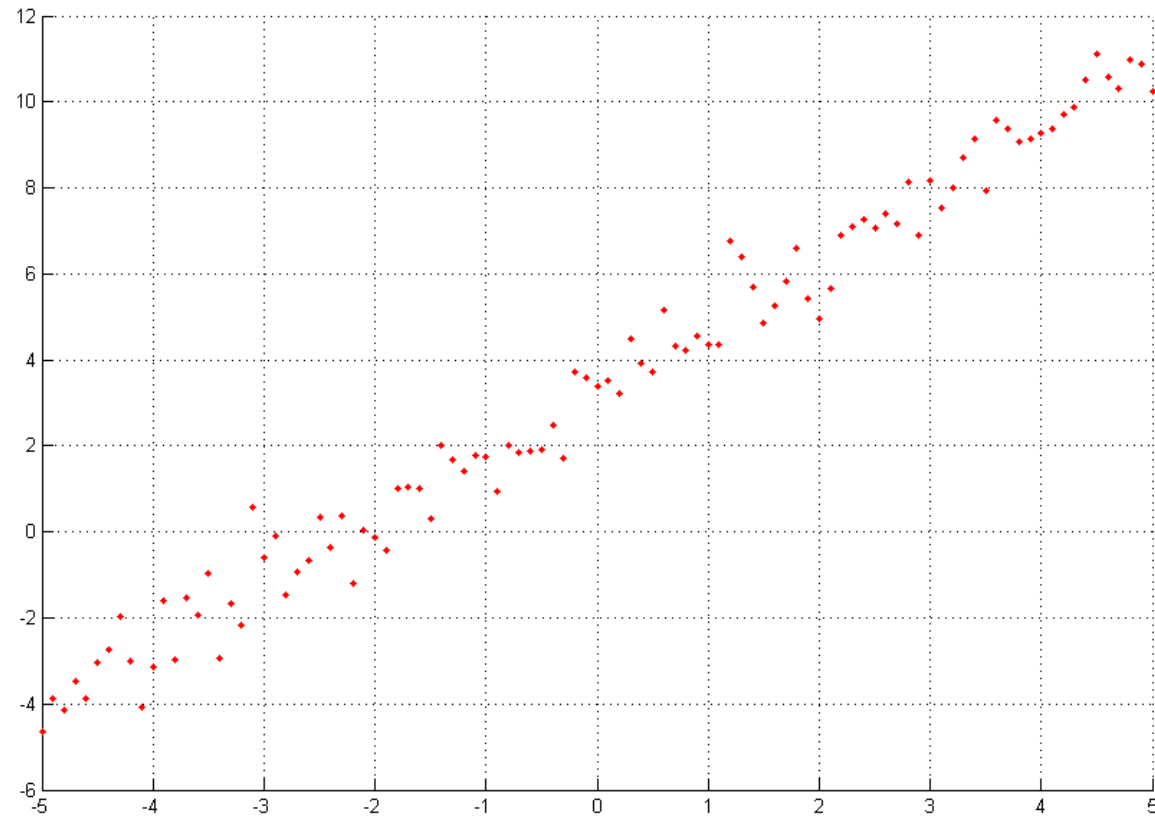
```
up = B.triangularView<Eigen::Upper>();
```

```
cout << up << endl << endl << low << endl;
```

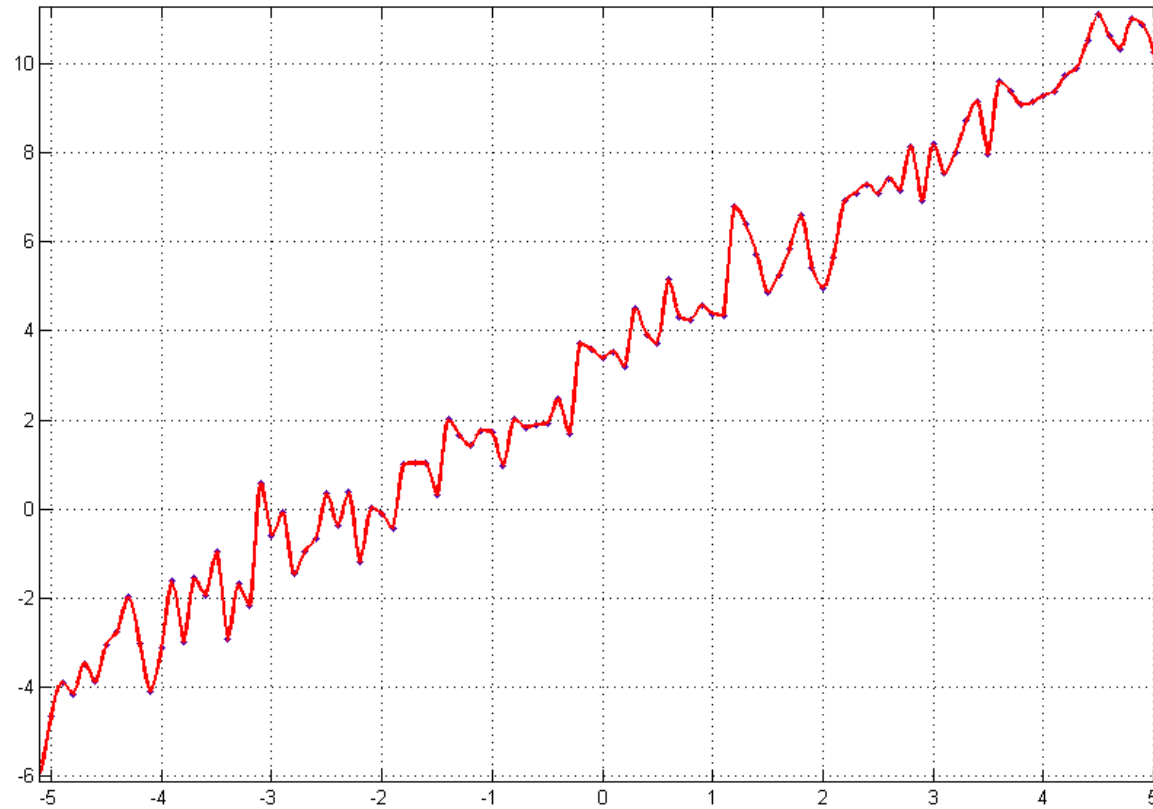
Метод Зейделя

$$x_{n+1} = \tilde{A}^{-1}B_l x_{n+1} + \tilde{A}^{-1}B_u x_n + \tilde{A}^{-1}\tilde{b}$$

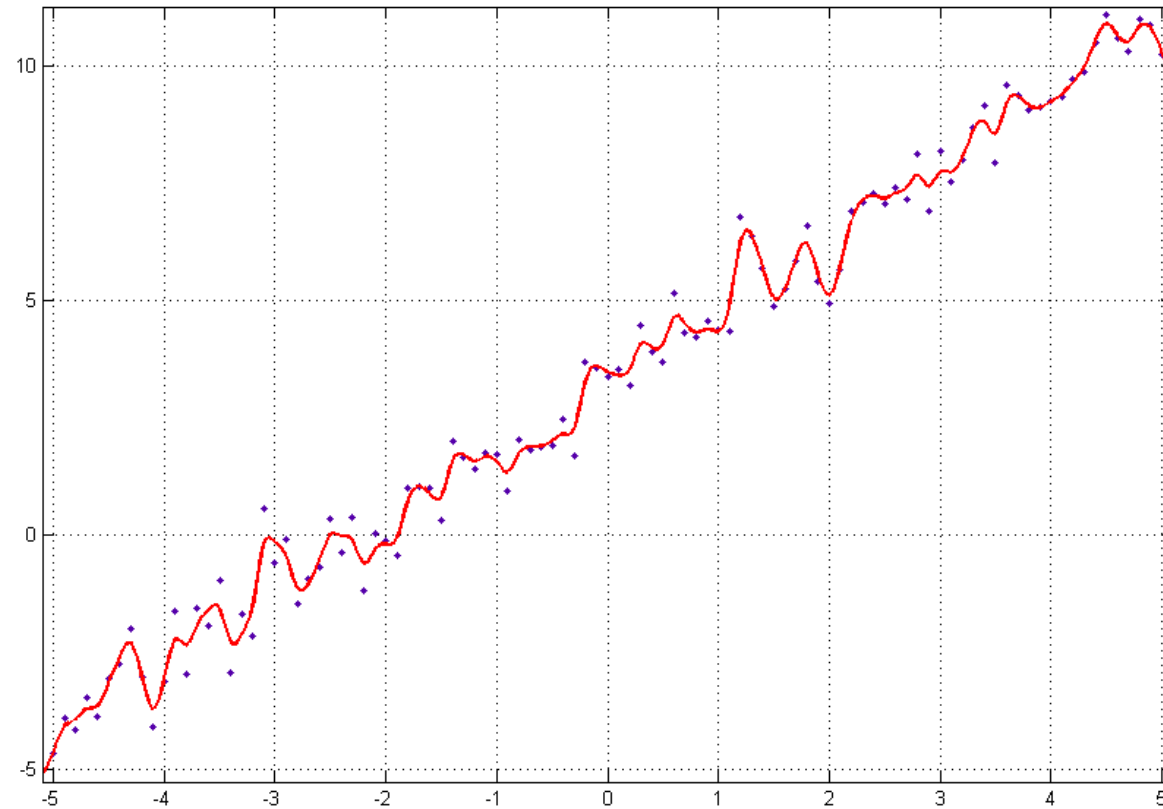
Обработка «зашумлённых данных»



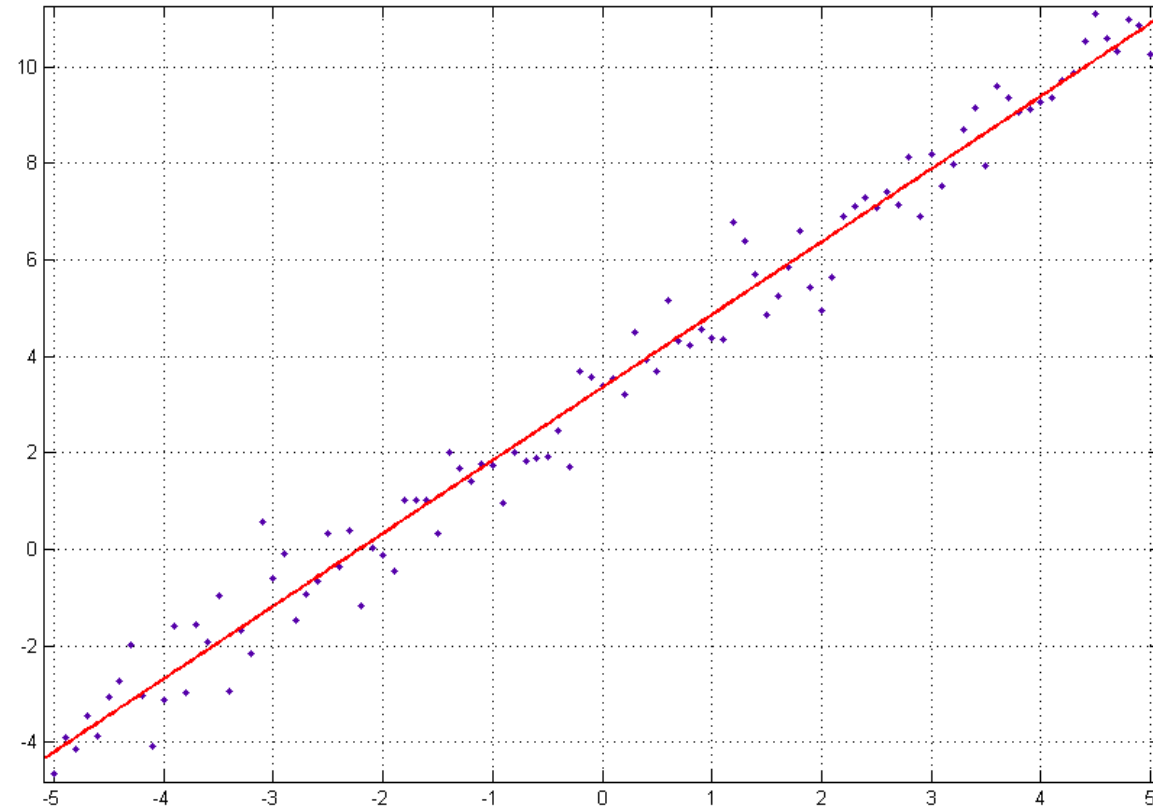
Обработка «зашумлённых данных»



Обработка «зашумлённых данных»



Обработка «зашумлённых данных»



Обработка «зашумлённых данных»

Как выбрать правильное приближение?

Как оценить его параметры?

Как понять, что получилось?

МНК-оценка

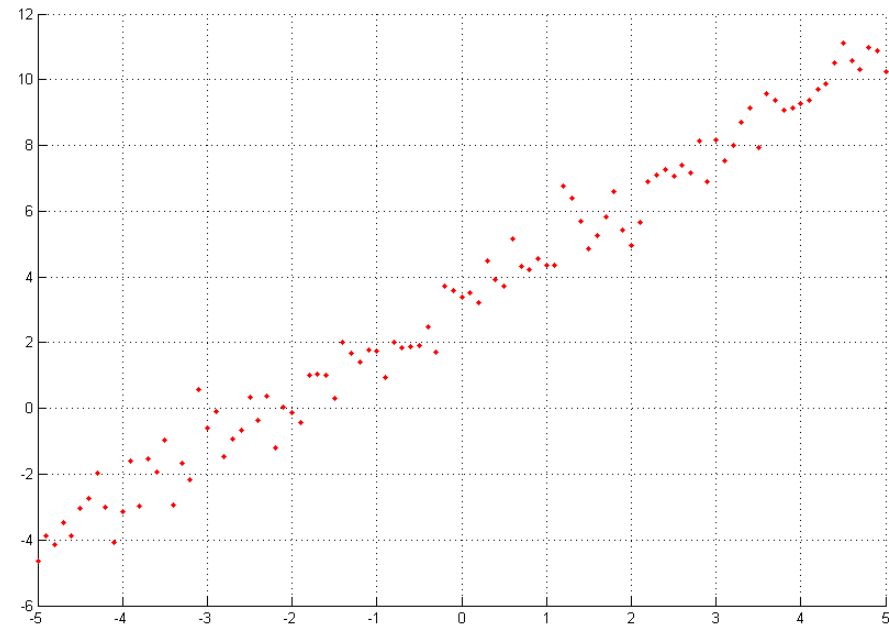
Используется для линейных по параметрам моделей

Подразумевает, что модель мы каким-то образом уже выбрали

Например:

x – данные

y – значения



МНК-оценка

Модели могут быть разными:

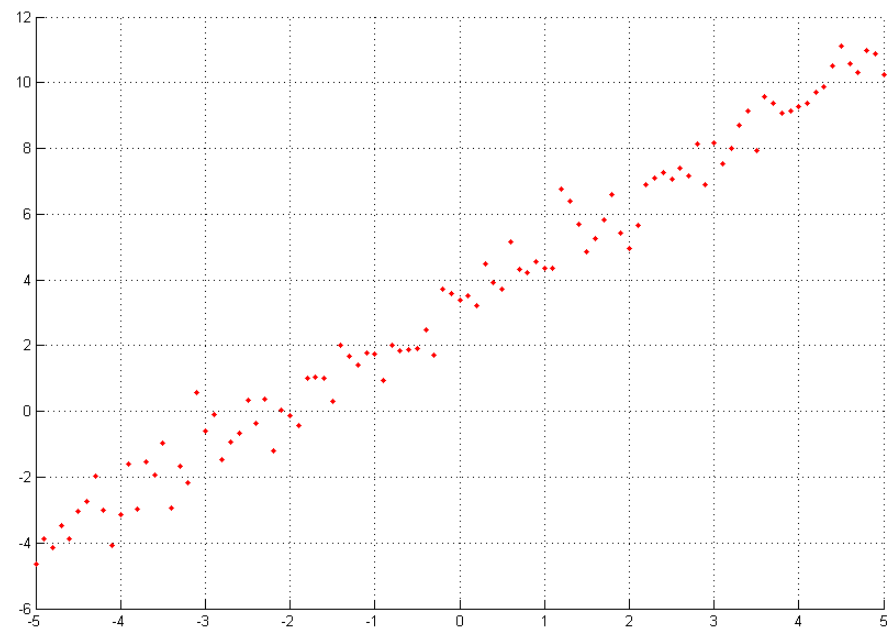
$$u = C = \text{const}$$

$$u = A \cdot x + B$$

$$u = A \cdot x^2 + B \cdot x + C$$

$$u = A \cdot e^{Bx}$$

$$u = C_1 \cos(\omega_1 x) + C_1 \sin(\omega_1 x)$$



МНК-оценка

Выберем линейный полином:

$$u = A \cdot x + B$$

- Получим набор значений $u_i = A \cdot x_i + B$
- При этом у нас есть y_i - практически полученные результаты
- Очевидно, что в идеальном случае $u_i = y_i$
- В реальности $r_i = u_i - y_i$ - невязка
- Нужно выбрать A и B так, чтобы невязка была минимальной

МНК-оценка

Будем минимизировать квадратичный функционал:

$$\sum_i r_i^2 \rightarrow \min$$

Можно записать через вектор:

$$r^T r \rightarrow \min$$

Условие минимума – равенство нулю первой вариации:

$$\delta r^T r = 0$$

МНК-оценка

Запишем уравнения для компонент r_i :

$$r_i = x_i \cdot A + 1 \cdot B - y_i$$

Можно записать матричное уравнение:

$$\begin{pmatrix} \dots \\ r_i \\ \dots \end{pmatrix} = \begin{pmatrix} \dots & \dots & \dots \\ x_i & 1 & y_i \\ \dots & \dots & \dots \end{pmatrix} \begin{pmatrix} A \\ B \\ -1 \end{pmatrix}$$

Тогда функционал можно записать как

$$(A \quad B \quad -1) \begin{pmatrix} \dots & x_i & \dots \\ \dots & 1 & \dots \\ \dots & y_i & \dots \end{pmatrix} \begin{pmatrix} \dots & \dots & \dots \\ x_i & 1 & y_i \\ \dots & \dots & \dots \end{pmatrix} \begin{pmatrix} A \\ B \\ -1 \end{pmatrix} \rightarrow \min$$

После этого первую вариацию можно записать как

$$\begin{pmatrix} \dots & x_i & \dots \\ \dots & 1 & \dots \\ \dots & y_i & \dots \end{pmatrix} \begin{pmatrix} \dots & \dots & \dots \\ x_i & 1 & y_i \\ \dots & \dots & \dots \end{pmatrix} \begin{pmatrix} A \\ B \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

То же самое

$$F \cdot \begin{pmatrix} A \\ B \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

где $F = \begin{pmatrix} \dots & x_i & \dots \\ \dots & 1 & \dots \\ \dots & y_i & \dots \end{pmatrix} \begin{pmatrix} \dots & \dots & \dots \\ x_i & 1 & y_i \\ \dots & \dots & \dots \end{pmatrix}$ -матрица 3x3

МНК-оценка

У нас 3 уравнения, но всего 2 переменных

Можно преобразовать у уравнению 2x2:

$$\begin{pmatrix} \cdots & x_i & \cdots \\ \cdots & 1 & \cdots \end{pmatrix} \begin{pmatrix} \cdots & \cdots \\ x_i & 1 \\ \cdots & \cdots \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} \cdots & x_i & \cdots \\ \cdots & 1 & \cdots \end{pmatrix} \begin{pmatrix} \cdots \\ y_i \\ \cdots \end{pmatrix}$$

То же самое:

$$G^T G \begin{pmatrix} A \\ B \end{pmatrix} = G^T y$$

МНК-оценка

А если модель экспоненциальная?

$$u = A \cdot e^{Bx}$$

Можно получить линейную логарифмированием:

$$\ln u = \ln A + Bx$$

Задача

Решите систему уравнений, используя Eigen (или собственное решение)

$$\begin{pmatrix} 1 & -0.86576 & 0.49954 \\ 0.86576 & 0.50046 & -0.86576 \\ 0.49954 & 0.86576 & 1 \end{pmatrix} x = \begin{pmatrix} 0.63378 \\ 0.50046 \\ 2.3653 \end{pmatrix}$$